CS256

Chapter 15: Inheritance, Polymorphism, and Virtual Functions

Hanoi University of Science and Technology

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Eighth Edition, Pearson, 2015
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

Example

A graded activity can receive a numeric score such as 60, 70, 80, 90, and so on, and a letter grade such as A, B, C, D, or F.

- `GradedActivity.h`
- `GradedActivity.cpp`
- `main.cpp`

Example GradedActivity.h

```
#ifndef GRADEDACTIVITY_H
#define GRADEDACTIVITY_H

// GradedActivity class declaration
class GradedActivity
{
private:
    double score; // To hold the numeric score
public:
    GradedActivity() { score = 0.0; } // Default constructor
    GradedActivity(double s) { score = s; } // Constructor

    void setScore(double s) { score = s; } // Mutator

    // Accessor functions
    double getScore() const { return score; }
    char getLetterGrade() const;
};
#endif
```

Example: GradedActivity.cpp

```
#include "GradedActivity.h"

char GradedActivity::getLetterGrade() const
{
    char letterGrade; // To hold the letter grade

    if (score > 89)
        letterGrade = 'A';
    else if (score > 79)
        letterGrade = 'B';
    else if (score > 69)
        letterGrade = 'C';
    else if (score > 59)
        letterGrade = 'D';
    else
        letterGrade = 'F';

    return letterGrade;
}
```

Example: main.cpp

```
// This program demonstrates the GradedActivity class.
#include <iostream>
#include "GradedActivity.h"
using namespace std;

int main()
{
    double testScore;  // To hold a test score
    GradedActivity test; // Create a GradedActivity object

    // Get a numeric test score from the user.
    cout << "Enter score: "; cin >> testScore;

    test.setScore(testScore);  // Store score.

    // Display the letter grade for the test.
    cout << "The grade for that test is "
         << test.getLetterGrade() << endl;
    return 0;
}
```

Example

- Many different types of graded activities: final exams, essays, ...
- Numeric scores might be determined differently for each of these graded activities
- We can create derived classes to handle each one.

What Is Inheritance?

- Provides a way to create a new class from an existing class.
- The new class is a specialized version of the existing class.
- Inheritance establishes an "is a" relationship between classes.
  - A poodle is a dog.
  - A car is a vehicle.
  - A flower is a plant.
  - A football player is an athlete.
- An object of a derived class 'is a(n)' object of the base class.
- A derived object has all of the characteristics of the base class.

Inheritance – Terminology and Notation

- Base class (or parent) – inherited from
- Derived class (or child) – inherits from the base class
- Notation:

```
class Student
{
    . . .
};
class UnderGrad : public student
{
    . . .
};
```

What Does a Child Have?

An object of the derived class has:

- all members defined in child class.
- all members declared in parent class.

An object of the derived class can use:

- all public members defined in child class.
- all public members defined in parent class.

What Does a Child Have?

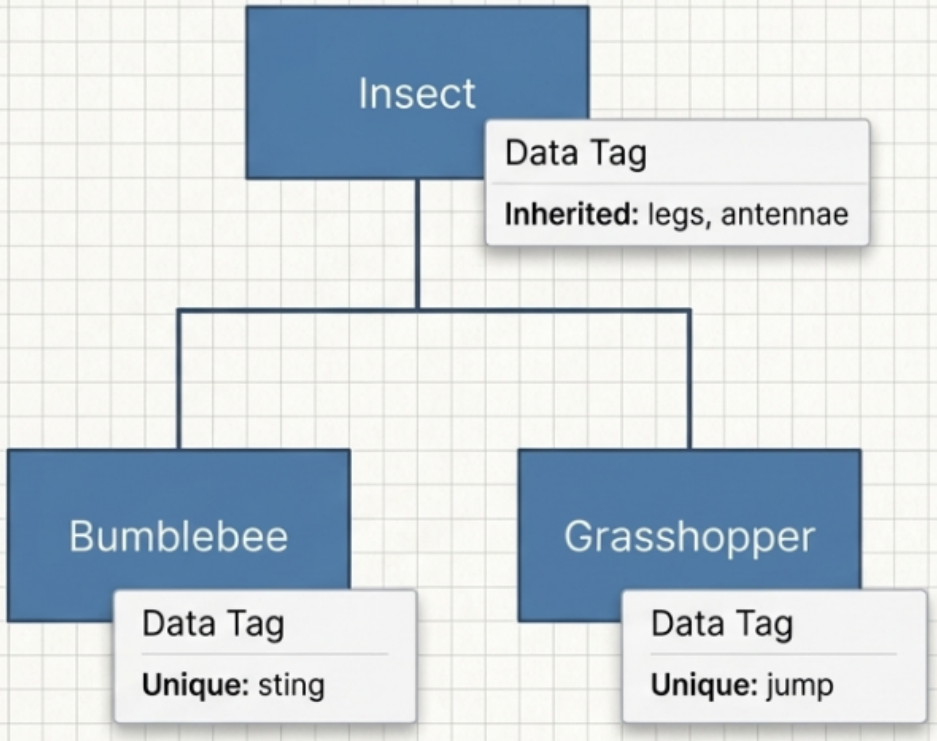
Concept

Inheritance creates a specialized version of an existing class.

An object of a derived class IS AN object of the base class.

Syntax

```
class Grasshopper : public Insect {
  ...
};
```



Example continues: FinalExam.h

```
#ifndef FINALEXAM_H
#define FINALEXAM_H
#include "GradedActivity.h"

class FinalExam : public GradedActivity
{
private:
    int numQuestions;     // Number of questions
    double pointsEach;    // Points for each question
    int numMissed;        // Number of questions missed
public:
    // Default constructor
    FinalExam()
        { numQuestions = 0;
          pointsEach = 0.0;
          numMissed = 0; }

    // Constructor
    FinalExam(int questions, int missed)
        { set(questions, missed); }
```

Example continues: FinalExam.h

```
    // Mutator function
    void set(int, int);     // Defined in FinalExam.cpp

    // Accessor functions
    double getNumQuestions() const
        { return numQuestions; }

    double getPointsEach() const
        { return pointsEach; }

    int getNumMissed() const
        { return numMissed; }
};
#endif
```

Example continues: FinalExam.cpp

```
#include "FinalExam.h"
// The parameters are the number of questions and the
void FinalExam::set(int questions, int missed)
{
    double numericScore;  // To hold the numeric score

    // Set the number of questions and number missed.
    numQuestions = questions;
    numMissed = missed;

    // Calculate the points for each question.
    pointsEach = 100.0 / numQuestions;

    // Calculate the numeric score for this exam.
    numericScore = 100.0 - (missed * pointsEach);

    // Call the inherited setScore function to set
    // the numeric score.
    setScore(numericScore);
}
```

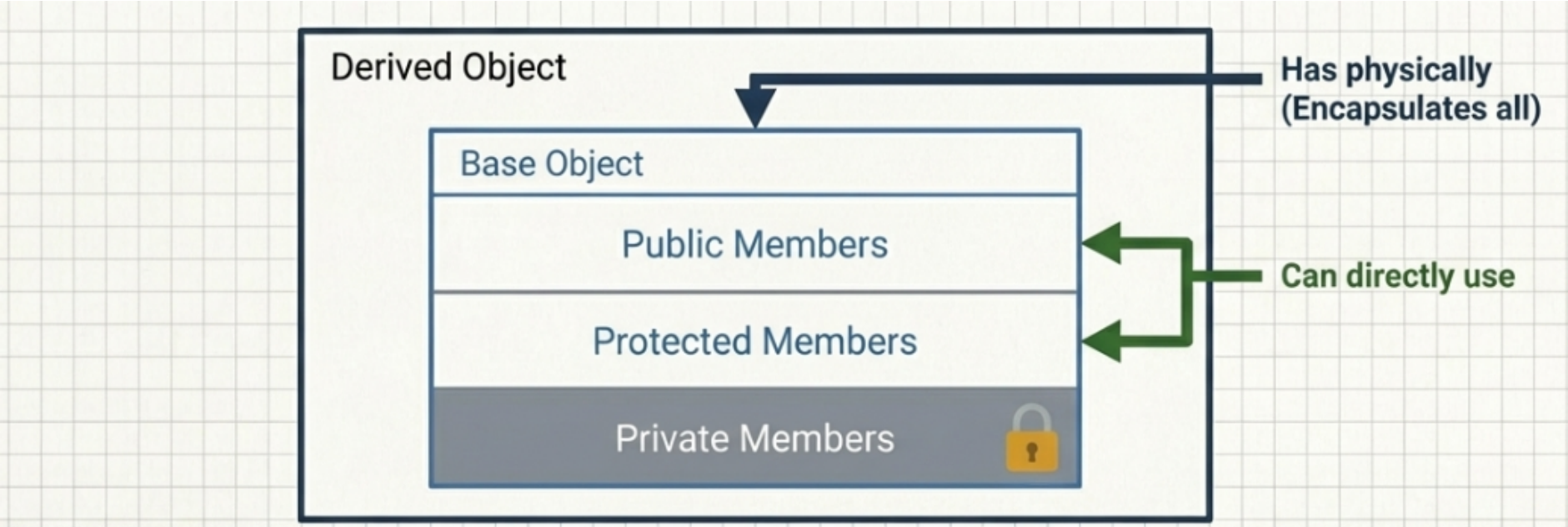
Example continues: main.cpp

```
// This program demonstrates a base class and a derived class.
#include <iostream>
#include <iomanip>
#include "FinalExam.h"
using namespace std;
int main()
{
    int questions, missed;
    cout << "Enter nb of questions"; cin >> questions;
    cout << "Enter nb of questions missed? "; cin >> missed;

    FinalExam test(questions, missed);

    cout << setprecision(2);
    cout << "\nEach question counts " << test.getPointsEach()
         << " points.\n";
    cout << "The score is " << test.getScore() << endl;
    cout << "The grade is " << test.getLetterGrade() << endl;
    return 0;
}
```

Protected Members and Class Access

- `protected` member: like `private`, but accessible by objects of derived class.
- `public`: object of derived class can be treated as object of base class (not vice-versa)
- `protected`: more restrictive than public, but allows derived classes to know details of parents
- `private`: prevents objects of derived class from being treated as objects of base class

Protected Members and Class Access



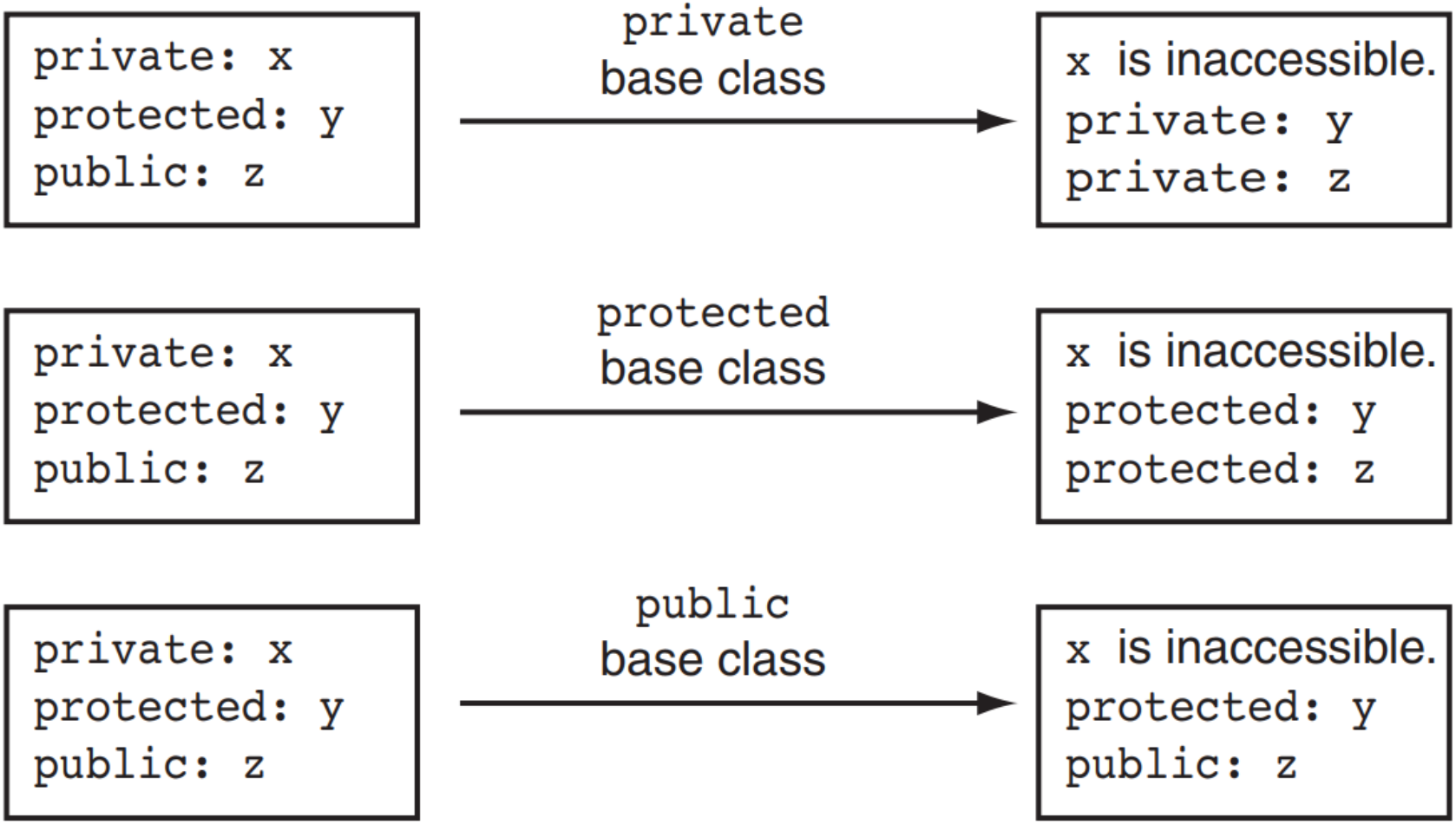
Protected Members and Class Access

Inheritance Access Specifier	Base Class Member Type: Public	Base Class Member Type: Protected	Base Class Member Type: Private
Public Inheritance	Public	Protected	Inaccessible
Protected Inheritance	Protected	Protected	Inaccessible
Private Inheritance	Private	Private	Inaccessible

Context Note

Class access specification acts as a filter, determining how the parent's members manifest within the child's architecture.
Private members of the base **class** are **always** inaccessible directly from the derived class, regardless of the inheritance type.

Protected Members and Class Access



Protected Members and Class Access

class Grade

```
private members:
   char letter;
   float score;
   void calcGrade();
public members:
   void setScore(float);
   float getScore();
   char getLetter();
```

class Test : public Grade

```
private members:
   int numQuestions;
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
```

When `Test` class inherits from `Grade` class using `public` class access, it looks like this: →

```
private members:
   int numQuestions;
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
   void setScore(float);
   float getScore();
   float getLetter();
```

Protected Members and Class Access

class Grade

```
private members:
   char letter;
   float score;
   void calcGrade();
public members:
   void setScore(float);
   float getScore();
   char getLetter();
```

class Test : protected Grade

```
private members:
   int numQuestions;
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
```

When `Test` class inherits from `Grade` class using `protected` class access, it looks like this: →

```
private members:
   int numQuestions:
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
protected members:
   void setScore(float);
   float getScore();
   float getLetter();
```

Protected Members and Class Access

class Grade

```
private members:
   char letter;
   float score;
   void calcGrade();
public members:
   void setScore(float);
   float getScore();
   char getLetter();
```

class Test : private Grade

```
private members:
   int numQuestions;
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
```

When `Test` class inherits from `Grade` class using `private` class access, it looks like this: →

```
private members:
   int numQuestions;
   float pointsEach;
   int numMissed;
   void setScore(float);
   float getScore();
   float getLetter();
public members:
   Test(int, int);
```

Constructors and Destructors

- Derived classes can have their own constructors and destructors
- When an object of a derived class is created, the base class's constructor is executed first, followed by the derived class's constructor
- When an object of a derived class is destroyed, its destructor is called first, then that of the base class

Constructors and Destructors

```
#include <iostream>
using namespace std;

class BaseClass
{
public:
    BaseClass()  // Constructor
        { cout << "This is the BaseClass constructor.\n"; }

    ~BaseClass() // Destructor
        { cout << "This is the BaseClass destructor.\n"; }
};
```

Constructors and Destructors

```
//*********************************
// DerivedClass declaration       *
//*********************************

class DerivedClass : public BaseClass
{
public:
    DerivedClass()  // Constructor
        {cout << "This is the DerivedClass constructor.\n";}

    ~DerivedClass() // Destructor
        {cout << "This is the DerivedClass destructor.\n";}
};
```

Constructors and Destructors

```
//*********************************
// main function                  *
//*********************************

int main()
{
    cout << "We will now define a DerivedClass object.\n";

    DerivedClass object;

    cout << "The program is now going to end.\n";
    return 0;
}
```

Redefining Base Class Functions

- **Redefining function**: function in a derived class that has the same name and parameter list as a function in the base class.
- Typically used to replace a function in base class with different actions in derived class.

Redefining Base Class Functions: CurvedActivity.h

```
#ifndef CURVEDACTIVITY_H
#define CURVEDACTIVITY_H
#include "GradedActivity.h"

class CurvedActivity : public GradedActivity
{
protected:
    double rawScore;        // Unadjusted score
    double percentage;      // Curve percentage
public:
    // Default constructor
    CurvedActivity() : GradedActivity()
        { rawScore = 0.0; percentage = 0.0; }

    // Mutator functions
    // Redefined setScore function
    void setScore(double s)
        { rawScore = s;
          GradedActivity::setScore(rawScore * percentage); }
```

Redefining Base Class Functions: CurvedActivity.h

```
void setPercentage(double c)
        { percentage = c; }

    // Accessor functions
    double getPercentage() const
        { return percentage; }

    double getRawScore() const
        { return rawScore; }
};
#endif
```

Redefining Base Class Functions: main.h

```
#include <iostream>
#include <iomanip>
#include "CurvedActivity.h"
using namespace std;

int main()
{
    double numericScore;  // To hold the numeric score
    double percentage;    // To hold curve percentage

    // Define a CurvedActivity object.
    CurvedActivity exam;

    // Get the unadjusted score.
    cout << "Enter the student's raw numeric score: ";
    cin >> numericScore;

    // Get the curve percentage.
    cout << "Enter the curve percentage for this student: ";
    cin >> percentage;
```

Redefining Base Class Functions: main.h

```
    // Send the values to the exam object.
    exam.setPercentage(percentage);
    exam.setScore(numericScore);

    // Display the grade data.
    cout << fixed << setprecision(2);
    cout << "The raw score is "
         << exam.getRawScore() << endl;
    cout << "The curved score is "
         << exam.getScore() << endl;
    cout << "The curved grade is "
         << exam.getLetterGrade() << endl;

    return 0;
}
```

Class Hierarchies

Class C inherits class B, Class B inherits class A.